What is Halftoning

Objective:
- Render an image with multiple levels of grey or color on a device with fewer tones

Applications:
- Print a greyscale image on a bi-level device (laser printer)
- Display a true color image on a limited color display (LED billboard)

Thresholding

- Throw away the low-order bits of intensity information
  - If intensity $> threshold_value$
    - Display full intensity at this channel
  - Otherwise
    - Display zero intensity at this channel

Tradeoff between Spatial & Tonal Resolutions

- $m \times n$ halftone cell provide $m \times n + 1$ tone levels
- Thresholding:
  - full resolution, 2 levels
  - $2 \times 2$ cell: 1/2 resolution, 5 levels
  - $4 \times 4$ cell: 1/4 resolution, 17 levels

Pixel Patterns

- The overall intensity of a given cell is determined by the number of pixels that are turned on
  - The locations of the on-pixels do not matter
  - The layouts of on-pixels under different intensities is referred as pixel pattern
    - The desired pattern is application dependent
Requirements for Pixel Pattern I

- Minimize visual artifacts:
  - When multiple pixel pattern are displayed together, there should be as fewer noticeable artifacts as possible
  - Avoid spatial patterns

Ordered Dither

- Basic idea:
  - Instead of deciding which pixels to turn on within each cell, use a threshold matrix to perform thresholding
- Values in threshold matrix are:
  - Ordered & scaled by maximum image intensity
  - Not affected by intensities of pixels

Dispersed-Dot Dither

- On-pixel are scattered evenly on screen
  - For device that handle individual dots well
  - Optimal pattern:
    - The number of dots are even in submatrix

Generate Dispersed-Dot Matrix

\[
\begin{pmatrix}
0 & 2 \\
3 & 1 \\
\end{pmatrix}
\]

\[
D^{(2)} = \begin{pmatrix}
0 & 8 & 2 & 10 \\
12 & 4 & 14 & 6 \\
3 & 11 & 1 & 9 \\
15 & 7 & 13 & 5 \\
\end{pmatrix}
\]

\[
D^{(2a)} = \begin{pmatrix}
4D^{(2)} & 2U^{(a)} \\
4D^{(2)} + 2U^{(a)} & 4D^{(2)} + U^{(a)} \\
\end{pmatrix}
\]

Ordered Dither Algorithm

// Input: Image A with value between 0~Imax.
// Dither matrix D of size n x n.
// Output: Matrix B with value 0 & 1.

for (int j = 0; j < height; j++)
  for (int i = 0; i < width; i++) {
    int p = i % n;
    int q = j % n;
    if (A[i][j] > D[p][q])
      B[i][j] = 1;
    else
      B[i][j] = 0;
  }

Generate Dispersed-Dot Matrix
Clustered-Dot Dither
- On-pixels are adjacent to each other
- Increasing intensity -> increasing dot size

Orientation of Dot Patterns
- Directly using the aforementioned dither matrix yields horizontal dots array
- Human eye is good at discerning horizontal and vertical lines
- Avoid using 0° or 90° directions

Dither Matrix for Diagonal Direction
- To produce dots along diagonal direction, the dither matrix need to have a diamond shape
- For easy handling, square shaped matrix is used for generating 2 dots at the same time

Generate Clustered-Dot Matrix

Error Diffusion
- Proposed by Floyd & Steinberg in 1975
- Basic idea:
  - Measure the error at current pixel caused by thresholding
  - Distribute the error to the neighbors for compensation
  - The global error is minimized

Error Diffuse Direction
- Only propagate errors to unprocessed pixels
- Processing order is top-left to bottom-right
- Weights are set for integer only calculation
- Closer pixels receive more error
- Soon to process pixels receive more error
Error Diffusion Algorithm

```java
for (int j=0 ; j<height ; j++)
    for (int i=0 ; i<width ; i++) {
        if ( A[i][j] < 0.5 )
            B[i][j] = 0;
        else
            B[i][j] = 1;
        error = A[i][j] - B[i][j];
        A[i+1][j] += \alpha \times error;
        A[i-1][j+1] += \beta \times error;
        A[i][j+1] += \gamma \times error;
        A[i+1][j+1] += \delta \times error;
    }
```